Задание: Паттерн Состояние.

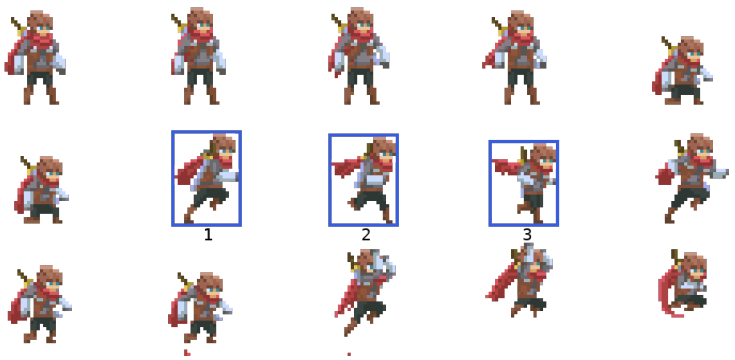
Описание программы

В папке `src` лежит исходный код программы, в которой используется паттерн Состояние для описания передвижения персонажа двумерной компьютерной игры. Персонаж может передвигаться по карте, при этом он может находиться в следующих состояниях:

- `Idle` – стояние на месте
- `Running` – бег
- `Falling` – обычное падение. При прыжке (клавиша `Space`) персонаж переходит в это состояние.
- `Sliding` – скольжение. В это состояние можно перейти из состояния бега, нажав на клавишу левый `Shift`. По истечению некоторого времени, при условии, что персонаж касается земли, он переходит в состояние `Idle` или `Running`.
- `Hooked` – зацепление за край блока. В это состояние можно перейти из состоя `Falling` оказавись рядом с краем блока. Выйти из этого состояния можно подпрыгнув или нажав стрелочку вниз.

Класс `Animation`

В программе используется спрайтовая анимация. Все возможные положения персонажа во всех состояниях хранятся в одном изображении `hero.png`. Эта текстура загружается в программу с помощью класса `sf::Texture`. В дальнейшем мы можем отрисовывать на экране любой прямоугольник из этой текстуры. Для удобства отрисовки этих прямоугольников на экран используется специальный объект класса `sf::Sprite`.



Например, для отрисовки бега персонажа сначала на месте персонажа рисуется изображение из прямоугольника `1` (смотрите рисунок), затем изображение из прямоугольника `2` и так далее.

Поля класса `Animation`:

- `mTextureRects` – контейнер, который хранит в себе координаты всех прямоугольников данной анимации.
- `mCurrentFrame` – номер текущего кадра-прямоугольника из вектора `mTextureRects`.
- `mAnimationSpeed` – скорость анимации
- `mTime` – текущее время анимации
- `mType` – тип анимации. Может принимать следующие значения:
  - `Animation::Repeat` – анимация повторяется. Используется, например, для анимации бега персонажа.
  - `Animation::OneIteration` – анимация проигрывается один раз, а затем застывается на последнем кадре. Используется для анимации скольжения и анимации зацепления за отвес.

Методы класса `Animation`:

- `void update(float dt)` – Увеличивает время текущей анимации на `dt`. Может изменить текущий кадр анимации. Вызывается каждый кадр.
- `void updateSprite(sf::Sprite& sprite)` – Устанавливает соответствующий прямоугольник текстуры для передаваемого спрайта.

Класс `Player`

Поля класса `Player`:

- `mPosition` – положение персонажа
- `mVelocity` – скорость персонажа
- `mCollisionRect` – прямоугольник, используемый для обнаружения столкновений персонажа с миром. Координаты прямоугольника отсчитываются от положения персонажа (`mPosition`).
- `mIsColliding` – переменная, которая показывает касается ли перонаж какого либо из блоков.
- `mpState` – указатель на состояние персонажа. Тут используется полиморфизм, чтобы менять состояние персонажа.
- `mTexture` – текстура
- `mSprite` – спрайт
- `mScaleFactor` – масштаб персонажа.
- `mIsFacedRight` – параметр, задающий направление персонажа.

Методы класса `Player`:

- `void update(float dt)` – обновляем положение персонажа и анимацию на время `dt` вперёд.
- `void draw(sf::RenderWindow& window)` – рисуем персонажа на окне.
- `void handleEvents(const sf::Event& event)` – обрабатываем все события, связанные с персонажем.
- `bool handleCollision(const sf::FloatRect& rect)` – проверяем столкновение персонажа и прямоугольника `rect`.
- `void handleAllCollisions(const std::vector<sf::FloatRect>& blocks)` – проверяем столкновение персонажа с вектором из прямоугольников. Устанавливаем значение поля `mIsColliding`.

Класс `World`

Поля класса `World`:

- `mBlocks` – контейнер-вектор, который хранит в себе прямоугольки, описывающие карту мира.
- `mPlayer` – игрок.
- `mGravity` – сила гравитации.
- `mView` – вид. Объект специального класса `sf::View`, который позволяет удобно двигать камеру в 2-х измерениях.

Методы класса `World`:

- `void update(float dt)` – обновляем мир на `dt` вперёд по времени.
- `void setView()` – устанавливаем камеру в зависимости от положения игрока.
- `draw`, `handleEvents`, `addBlock`.

Класс `PlayerState` и его наследники

Состояния персонажа реализовано с помощью абстрактного класса `PlayerState` и его наследников – классов описывающих каждое состояние: `Idle`, `Running`, `Falling`, `Sliding`, `Hooked`.

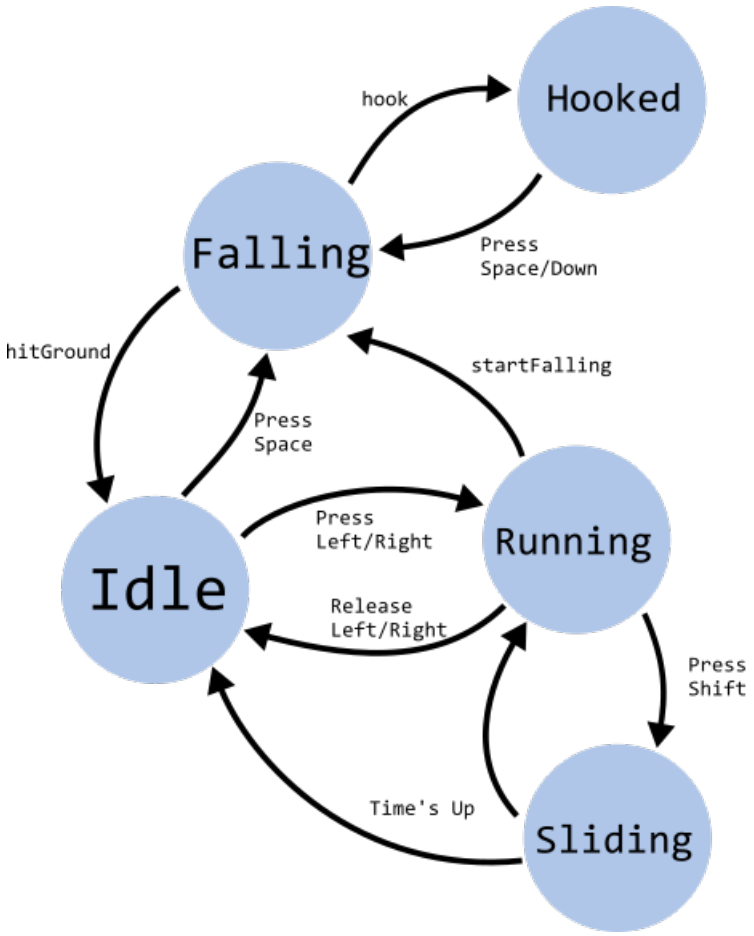
`PlayerState` хранит в себе анимацию(`mAnimation`) и силу прыжка (`kJumpingVelocity`) так как эти поля являются общими для всех классов. У этого класса также есть следующие абстрактные методы:

- `update` - метод, который вызывается каждый кадр.
- `handleEvents` - метод, для обработки всех событий (например, нажатий клавиш). Срабатывает каждый кадр.
- `startFalling` - метод, который срабатывает, когда персонаж перестаёт касаться земли. В этом случае, если мы находимся в состоянии `Running` или `Idle`, мы должны перейти в состояние `Falling`. В остальных случаях мы должны ничего не делать.
- `hitGround` - метод, который срабатывает, когда персонаж касается земли. В этом случае, если мы находимся в состоянии `Falling`, мы должны перейти в состояние `Idle`. В остальных случаях мы должны ничего не делать.
- `hook` - метод, который срабатывает, когда персонаж подходит близко к уступу. В этом случае, если мы находимся в состоянии `Falling`, мы должны перейти в состояние `Hooked`. В остальных случаях мы должны ничего не делать.

Эти абстрактные методы переопределяются в наследниках класса `PlayerState`. Благодаря такому полиморфизму и добивается удобное изменение состояний объекта. Так как указатель на состояние `mpState`, хранящийся в объекте класса `Player` может менять свой динамический тип и, соответственно, при вызове

```
mpState->update(dt);
```

будет вызываться переопределённый виртуальный метод, соответствующий текущему состоянию.



Задачи

Двойной прыжок

Измените состояние `Falling`, добавив возможность совершения персонажем двойного прыжка. Если персонаж перешёл в состояние `Falling` прыгнув или упав с уступа он должен иметь возможность прыгнуть в воздухе ещё один раз. Сила второго прыжка должна быть меньше, чем сила первого.

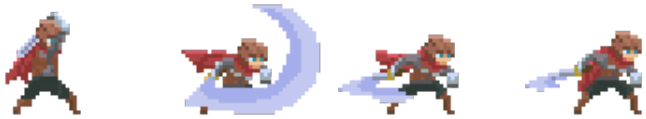
Состояние сидения

Добавьте новое состояние под названием `Sitting`. Персонаж должен переходить в это состояние из состояние `Idle` путём нажатия клавиши левый `Shift`.

Состояние атаки

Добавьте новые состояния для атаки персонажа. При нажатии на клавишу `X`, если персонаж находится в состояниях `Idle` или `Running`, он должен начать атаку и перейти в состояние `FirstAttack`.

После завершения первой атаки, если пользователь не нажимает клавишу `X`, то персонаж должен перейти в состояние `Idle`. Если же во время первой атаки пользователь ещё раз нажал клавишу `X`, то персонаж, по окончанию анимации, должен перейти из состояния `FirstAttack` в состояние `SecondAttack`.



После завершения второй атаки, если пользователь не нажимает клавишу `X`, то персонаж должен перейти в состояние `Idle`. Если же во время второй атаки пользователь ещё раз нажал клавишу `X`, то персонаж, по окончанию анимации, должен перейти из состояния `SecondAttack` в состояние `ThirdAttack`.

По окончанию третьей атаки персонаж должен перейти в состояние `Idle`. Если персонаж на момент начала атаки находился в состоянии `Running`, то он не должен мгновенно терять скорость. Скорость персонажа, полученная в состоянии `Running` должна плавно затухать во время проведения атак.

Анимацию атак можно посмотреть в файле `attack.gif`.

Разрушаемые объекты

Добавьте на карту объекты, которые можно будет разрушать. В качестве объектов можно взять просто круги, при разрушении они должны исчезать. Объекты должны разрушаться только в момент проведения атаки и только если они касаются или оказываются близко к мечу персонажа.